ALUPAS: Avaliação de desempenho e consumo de energia de softwares para sistemas embarcados

Bruno Nogueira [1]
Paulo Maciel [1]
Gustavo Callou [1]
Ermeson Andrade [1]
Eduardo Tavares [1]

Resumo: Com a proliferação de equipamentos portáteis operados por baterias, o projeto de sistemas embarcados de baixo consumo de energia tem despertado muito interesse nos últimos anos. Para atender aos requisitos de baixo consumo de energia, é essencial, ainda nas fases iniciais de desenvolvimento, dispor de mecanismos que auxiliem de forma rápida e exata a análise de possíveis alternativas de projeto. Este trabalho apresenta ALUPAS, um simulador estocástico baseado nas Redes de Petri Coloridas (CPN) para estimar o desempenho e consumo de energia de softwares para sistemas embarcados. Resultados experimentais mostram uma exatidão, em média, de 94% utilizando o simulador proposto em comparação aos valores reais medidos no hardware.

Abstract: Due to the widespread expansion of battery operated mobile devices, the low power design of embedded system has gained close attention in recent years. In order to attend low power requirements, it is essential, even in earlier phases of design, to count on fast and accurate mechanisms to help the analysis of possible design alternatives. This work presents ALUPAS, a stochastic simulator based on Coloured Petri Nets (CPN) for performance and energy consumption estimation of embedded systems applications. Experiments results have demonstrated an average accuracy of 94% using the proposed simulator in comparison with the real values measured on hardware.

1 Introdução

Sistema embarcado, ou embutido, é qualquer sistema de computação integrado a um sistema maior, com o objetivo de aumentar e/ou otimizar funcionalidades. Exemplos destes sistemas podem ser encontrados nos celulares, microondas, alarmes e *players* de mp3. Tipicamente estes sistemas são de tempo-real e apresentam uma série de restrições tais como: baixa capacidade de memória, fonte de energia, baixo processamento, dentre outras.

[1]Universidade Federal de Pernambuco, UFPE, Caixa Postal 7851
{bcsn,prmm,grac,ecda,eagt@cin.ufpe.br}

Com a proliferação de equipamentos portáteis operados por baterias, o projeto de sistemas embarcados de baixo consumo de energia tem despertado muito interesse nos últimos anos. O maior problema é que a evolução da tecnologia das baterias não vem acompanhado a crescente demanda por maior autonomia e mais desempenho nestes dispositivos [7]. Além dos aspectos relacionados à demanda do mercado, a própria evolução das tecnologias dos sistemas semicondutores, em função da maior funcionalidade que se coloca por unidade de área dos circuitos integrados, tem levado ao crescimento constante da potência dissipada destes sistemas. Uma vez que grande parte das funcionalidades do projeto de sistemas embarcados é implementada por *software*, o consumo de energia devido ao *software* pode ter uma grande influência no consumo total do projeto [16]. Isso faz com que ainda nas fases iniciais de projeto, seja essencial analisar de forma rápida e exata este consumo a fim de otimizar e, também, reduzir os riscos do projeto vir a ser mal sucedido. Além dos aspectos relacionados ao consumo de energia, previsibilidade temporal é uma importante questão em projetos de tempo-real.

Sem perda de generalização, os métodos de estimativa do consumo de energia podem ser classificados em: (i) Um método no qual as estimativas são realizadas através de um modelo do *hardware* da arquitetura alvo, e (ii) um segundo método no qual as estimativas são feitas a partir de medições no próprio *hardware* [12]. A exatidão das análises feitas através do primeiro método depende da exatidão do modelo utilizado. No entanto, é difícil construir um modelo com razoável exatidão devido ao aumento da complexidade das arquiteturas atuais. Outra dificuldade é a necessidade de se ter acesso a detalhes da tecnologia de implementação do *hardware* que constituem, normalmente, propriedades intelectuais. Através do segundo método as estimativas são mais exatas, pois os dados são gerados diretamente pelo próprio *hardware*. Apesar de existirem vários trabalhos que utilizam tais métodos, até onde os autores conhecem, poucos utilizam uma abordagem formal para a simulação.

Este trabalho apresenta ALUPAS, um simulador com o objetivo de estimar o desempenho (tempo de execução) e consumo de energia de *softwares* para sistemas embarcados de tempo-real com restrições de consumo de energia. Um modelo formal de simulação baseado nas Redes de Petri Coloridas (CPN - *Coloured Petri Nets*) [8] foi adotado. Dado o código fonte (com anotações) do *software* a ser avaliado, ALUPAS gera um modelo CPN do código e simula estocasticamente tal modelo com a finalidade de estimar seu tempo de execução e consumo de energia. A escolha das CPN para modelagem se justifica por elas possuírem as seguintes características: (i) formalismo, através de um consolidado conjunto de regras; (ii) representação gráfica, facilitando a cognição do modelo; (iii) representação hierárquica, suportando modelos complexos; (iv) unir funcionalidades semelhantes as das linguagens de programação.

Este trabalho está organizado da seguinte forma: Seção 2 apresenta os trabalhos relacionados; Seção 3 introduz alguns conceitos de Redes de Petri Coloridas; Seção 4 aborda o

modelo formal da simulação; Seção 5 descreve como o processador alvo deste trabalho foi caracterizado; Seção 6 explica os parâmetros de simulação adotados e como o resultado da simulação é avaliado; Seção 7 introduz a metodologia de avaliação adotada e o simulador ALUPAS; Seção 8 exibe alguns experimentos e resultados para validação do ALUPAS; A Seção 9 conclui este artigo.

2 Trabalhos Relacionados

Ao longo dos últimos anos, várias metodologias para estimar o consumo de energia têm sido desenvolvidas, no entanto, poucas destas metodologias foram associadas a uma ferramenta. Algumas ferramentas utilizam a abordagem de simulação do *hardware*, na qual o simulador é baseado em uma descrição (modelo) do *hardware* do processador. Neste tipo de simulação, o modelo de consumo é construído com base no modelo matemático do circuito (mais baixo nível) e/ou descrições de mais alto nível (RTL- *Register Transfer Level*). QuickPower [11] e PowerMill [5] são exemplos de ferramentas a nível de circuito. SimplePower [6], Wattch [3] e PowerTimer [2] são ferramentas que necessitam de um modelo RTL da arquitetura alvo para possibilitar as estimativas de potência. Apesar da boa exatidão dos resultados apresentados, o baixo nível dos modelos demanda um grande esforço computacional. Nestas ferramentas o comportamento do processador é simulado ciclo a ciclo. Desta forma, pequenos trechos de código podem ser facilmente analisados, mas para programas maiores a simulação pode se tornar impraticável. Além disto, neste tipo de simulação é necessário ter acesso a descrições de baixo-nível da arquitetura, o que nem sempre é possível.

Outra forma de simulação consiste em caracterizar o modelo de consumo do processador de acordo com o consumo associado as suas instruções. Esse modelo alimenta um simulador de instruções e a análise é realizada com base nos padrões de saída desse simulador. A ferramenta JouleTrack [15] é um exemplo deste tipo de simulador. Outro exemplo, é a metodologia baseada nas CPN desenvolvida por [13], na qual a modelagem estocástica do conjunto de instruções de microcontroladores da família 8051 foi demonstrada. A metodologia apresentada permite que o projetista associe probabilidades às instruções condicionais para representar os mais diversos cenários de execução do *software*. As desvantagens do referido trabalho são que além de ter sido utilizado uma ferramenta de uso geral (CPN Tools [14]) para realizar as simulações, os modelos propostos ficaram muito complexos. Como conseqüência direta, o tempo de avaliação torna-se impraticável quando a metodologia é aplicada a projetos reais.

3 Redes de Petri Coloridas

As Redes de Petri consistem em uma ferramenta matemática e gráfica que permite modelar diferentes tipos de sistemas, tais como: paralelos, concorrentes, assíncronos e não-determinísticos. Desde o modelo inicial das Redes de Petri, várias extensões têm sido propostas para permitir descrições mais concisas e representar características não contempladas nos primeiros modelos. Algumas das deficiências dos primeiros modelos são que: (i) não há conceitos de dados, tornando os modelos excessivamente grandes, pois toda manipulação de dados precisa ser representada diretamente na estrutura da rede, e (ii) não há conceito de hierarquia, não sendo possível construir modelos grandes a partir de sub-modelos separados com interfaces bem definidas. Dentre as extensões aos primeiros modelos, destacam-se as redes de alto-nível de Jensen [8], chamadas Redes de Petri Coloridas (CPN).

Um modelo CPN é apresentado na Figura 1 com a finalidade de exemplificar uma solução formal para o clássico problema do jantar dos filósofos. Lugares são representados graficamente por elipses, transições por retângulos e arcos por setas. O modelo possui 3 lugares, 2 transições e 6 arcos (de entrada e saída) que realizam a conexão dos lugares às suas respectivas transições. Além disso, também é possível observar a presença anotações à esquerda da rede (*Declarations*) e ao lado dos lugares, transições e arcos.

Em um modelo CPN, cada lugar pode estar marcado com um ou mais tokens e cada token possui um valor associado a si. Este valor é chamado de cor. O estado do sistema, chamado de marcação do modelo CPN, é representado pela quantidade e valor (cor) dos tokens presentes em cada lugar. Na Figura 1, um filósofo pode estar comendo ou pensando. Esse comportamento é representado através dos lugares *Pensando* e *Comendo*. O número de talheres disponíveis é modelado pela presença de tokens no lugar *Talheres livres*. Ao lado de cada lugar existe uma inscrição que define qual o conjunto de cores que os tokens neste lugar podem ter. Este conjunto de cores é semelhante aos tipos das linguagens de programação. Os lugares *Pensando* e *Comendo* têm o conjunto de cores (*colset*) *FILOSOFO*, e o lugar *Talheres livres* tem *colset TALHER*. As ações e eventos do modelo são modelados através das transições. Um arco de entrada indica que a transição pode remover tokens de seu correspondente lugar de entrada, similarmente, um arco de saída indica que a transição pode adicionar tokens ao seu correspondente lugar de saída. Uma transição é dita habilitada a ocorrer quando: (i) ela possui um token do tipo apropriado em cada um de seus arcos de entrada e (ii) quando a expressão de guarda (expressão booleana) associada a transição é avaliada como verdadeira. No modelo da Figura 1, a função *Talheres* mapeia cada filósofo a dois talheres ao seu lado. Quando, por exemplo, a transição *Pega talheres* ocorre, é removido um token do lugar *Pensando* e dois do lugar *Talheres livres*, e em seguida, um token é adicionado no lugar *Comendo*.

A definição não-hierárquica das Redes de Petri Coloridas é uma 9-tupla $CPN =$

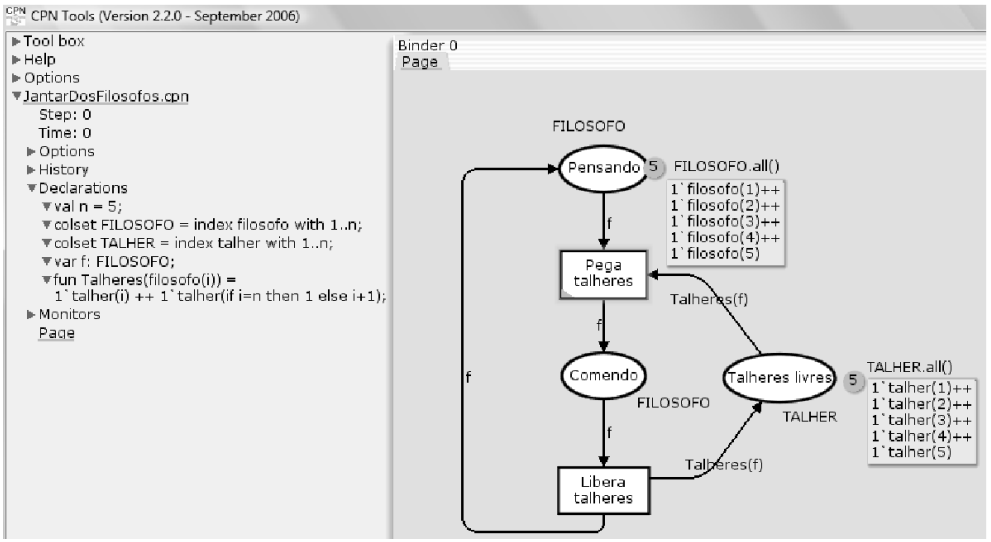

Figura 1. Problema do jantar dos filósofos em CPN.

$(\sum, P, T, A, N, C, G, E, I)$, onde $\sum$ é um conjunto finito de tipos não-vazios, chamado Conjunto de Cores; P é um conjunto finito de elementos, chamado Lugares; T é um conjunto finito de elementos, chamado Transições; A é um conjunto finito de arcos tal que $P \cap T = P \cap A = T \cap A =$, $N : A \rightarrow P \times T \rightarrow T \times P$ é uma função de nó; $C : P \rightarrow \sum$ é uma função de cor; G é uma função de guarda, que é definida a partir de T em expressões tais quais $\forall t \in T : [Tp(G(t)) = Bool \wedge Tp(Var(G(t))) \subseteq \sum]$; E é uma função de arco, definida a partir de A em expressões tais quais $\forall a \in A : [Tp(E(a)) = C(p(s))_{MS} \wedge Tp(Var(E(a))) \subseteq \sum]$, onde $p(a)$ é o lugar de $N(a)$ e C_{MS} denota o conjunto de todos os multi-conjuntos sobre C; I é a função de inicialização, definida a partir de P em expressões tais quais $\forall p \in P : [Tp(I(p)) = C(p(s))_{MS} \wedge Var(I(p)) = \emptyset]$, onde $Tp(expr)$ denota o tipo de uma expressão, $Var(expr)$ denota o conjunto de variáveis em uma expressão, $C(p)_{MS}$ denota o multi-conjunto sobre $C(p)$.

As CPN também permitem modelar estruturas hierárquicas. A idéia central é permitir a construção de modelos grandes através da composição de modelos menores. Estes modelos menores são chamados de páginas e são conectados uns aos outros através de lugares chamados de portas. Tais lugares podem ser do tipo porta de entrada ou porta de saída.

4 Modelo da simulação

No modelo proposto, o fluxo de controle do *software* é representado por um modelo CPN com dois níveis de hierarquia. No nível mais baixo estão os modelos básicos e no nível mais alto a composição destes modelos básicos. Os modelos básicos modelam as instruções do processador. Este trabalho tem como foco um microcontrolador da família ARM7, o Philips LPC2106 [1]. Quatro modelos básicos foram construídos para representar o conjunto de instruções do LPC2106: (i) *modelo de instrução ordinária*, (ii) *modelo de desvio condicional*, (iii) *modelo de desvio com retorno*, e (iv) *modelo de retorno para instruções de desvio com retorno*. Cada um destes modelos computa o consumo de energia e tempo de execução de uma instrução ou bloco de instruções durante a avaliação do modelo. Adicionalmente, estes modelos básicos foram alimentados com dados sobre o tempo de execução e consumo de energia das instruções do LPC2106, seguindo a metodologia de medições descrita na Seção 5.

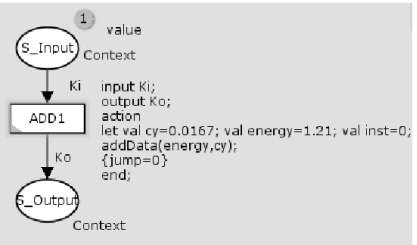

Figura 2. Modelo de instrução ordinária.

A Figura 2 exibe o *modelo de instrução ordinária*. Este modelo representa as instruções não condicionais do processador. Durante a simulação, este modelo computa o consumo de energia (*val energy*) e tempo de execução (*val cy*) através da função *addData(energy,cy)*. Os valores presentes em *energy* e *cy* são expressos em nanojoules (nJ) e microsegundos (μs), respectivamente. O mesmo modelo também é utilizado para representar um conjunto de instruções agrupadas em um único bloco básico depois que a rede sofre o processo de redução. O processo de redução consiste em transformar o modelo original, em um modelo mais simples, com as mesmas características e tempo de simulação menor. Um trecho de código com um único ponto de entrada e um único ponto de saída é candidato a se tornar um bloco básico.

A Figura 3 exibe o *modelo de desvio condicional*. Este modelo representa as instruções de desvio condicional do processador. É possível notar que os registradores do processador não são armazenados pelos modelos básicos. Ao invés de se comparar valores armazenados nos registradores, uma abordagem estocástica é adotada. Para determinar o fluxo de execução do código durante a simulação, são associadas probabilidades às instru-

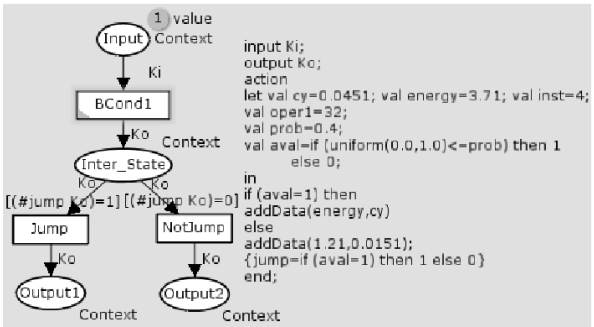

Figura 3. Modelo de desvio condicional.

ções condicionais. Como pode ser visto na Figura 3, durante a simulação do *modelo de desvio condicional*, ALUPAS gera um número aleatório entre 0 e 1 com distribuição uniforme e compara com a probabilidade associada (*val prob*). Caso o número aleatório seja menor ou igual a esta probabilidade, a condição de execução é avaliada como verdadeira e ocorre desvio no fluxo de execução, caso contrário não há desvio. Como será explicado na Seção 6, as probabilidades são definidas pelo projetista através de anotações no código.

Os modelos básicos *modelo de desvio com retorno*, e *modelo de retorno para instruções de desvio com retorno* são utilizados, respectivamente, para representar as instruções de chamada de função e de retorno de função do microcontrolador modelado.

Para o trecho de código exibido na Figura 4, a Figura 5 exibe o fluxo de controle no modelo proposto. Na Figura 5 cada transição é uma instância de um dos modelos básicos apresentados. Para automatizar a geração de modelos, um compilador, denominado Compilador Binário-CPN (ver Seção 7), foi criado e integrado ao ALUPAS. Este compilador recebe como entrada o código binário do *software* a ser avaliado e gera a composição dos modelos básicos CPN como saída. O processo de redução do modelo também é feito automaticamente pelo Compilador Binário-CPN.

5 Caracterização do processador

A medição do consumo de energia e tempo de execução de cada instrução é feita individualmente. Para obter os dados correspondentes a cada instrução, 10000 instâncias da instrução a ser medida são colocadas no corpo de um loop. Em seguida, os dados obtidos são divididos por 10000 para que seja obtido o custo individual de uma instrução.

O esquema de medição é exibido na exibido na Figura 6. Para medir o consumo de

```
1 FOR
2   MOV     r4, #1
3   B       TEST
4 LOOP
5   ADD     r4, r4, #1
6 TEST
7   CMP     r4,#0xa
8   BLT     LOOP
9   BX      r14
```

Figura 4. Trecho de código.

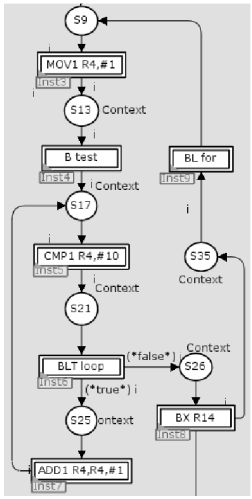

Figura 5. Modelo CPN do fluxo de controle: composição dos modelos básicos.

energia de uma instrução, um PC com a ferramenta AMALGHMA é conectado ao osciloscópio digital Agilent DSO03202A. Através do canal 2, o osciloscópio captura a corrente consumida medindo a queda de tensão em um resistor *shunt* colocado em série com o processador. O canal 1 é utilizado para indicar o início e o fim do intervalo a ser medido. A ferramenta AMALGHMA foi utilizada automatizar o processo de medição. Esta ferramenta adota um conjunto de métodos estatísticos, como bootstrap e método paramétrico, que são importantes no processo de medição devido a diversos fatores, como: (i) resolução do osciloscópio, e (ii) erro no resistor. É importante ressaltar que o processo de medições foi validado utilizando os dados relativos ao consumo de energia e tempo de execução presentes em [1] e no *datasheet* do LPC2106.

6 Parâmetros da simulação e análise dos resultados

Anotações no código fonte do *software* permitem que o projetista possa definir o cenário de execução a ser avaliado e o critério de parada da avaliação. A Seção 8.1 exibe um exemplo de código (assembly) anotado. As anotações da forma $<@\ Prob\ @>$ associam probabilidade *Prob* a uma instrução de desvio condicional. Assim, a probabilidade do desvio

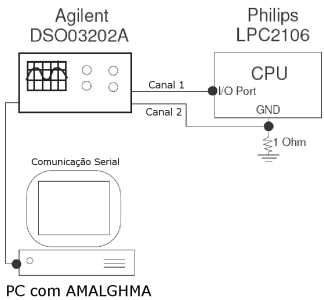

Figura 6. Esquema de medição.

ocorrer é igual à *Prob*. Este tipo anotação permite que o projetista avalie os mais diversos cenários de execução do *software*, bastando apenas modificar o valor destas probabilidades. As probabilidades de cada cenário podem ser obtidas por meio de: (i) inferências do projetista com base em seu conhecimento sobre o código e/ou (ii) análise do perfil de execução (*profiling*) do código utilizando ferramentas externas.

Devido a sua natureza estocástica, os resultados das simulações obviamente apresentam variações. Sendo assim, as estimativas geradas por ALUPAS não podem ser baseadas em apenas uma única simulação. Portanto, para que as estimativas sejam confiáveis, é necessário que a simulação seja replicada um certo número de vezes, e então utilizar os resultados de todas as replicações para conduzir as inferências. Duas questões que se colocam são: (i) o que fazer com os resultados das replicações? (ii) qual o número ideal de replicações para se obter estimativas confiáveis? A primeira pergunta é facilmente respondida utilizando a média amostral das replicações. O segundo problema é contornado gerando tantas replicações quantas forem necessárias até que um critério de parada seja satisfeito. O critério de parada adotado por ALUPAS se baseia no método de replicação independente (RI) [4].

Baseado na RI, a média amostral é definida através da realização de *b* replicações, em que cada replicação consiste em *m* simulações do código a ser avaliado. Assim, a média amostral da *i-ésima* replicação é dada por:

$$Z_i = \frac{1}{m} \sum_{j=1}^{m} Y_{i,j} \tag{1}$$

onde $Y_{i,j}$ é o resultado da simulação *j* da replicação *i*, para *i=1,2,...,b* e *j=1,2,...,m*.

Segue que a variância amostral é:

$$V_r = \frac{1}{b-1}\sum_{i=1}^{b}(Z_i - \overline{Z}_b)^2 \tag{2}$$

, em que a média amostral geral $\overline{Z}_b$ é dada por:

$$\overline{Z}_b = \frac{1}{b}\sum_{i=1}^{b} Z_i. \tag{3}$$

Utilizando o Teorema do Limite Central, pode-se mostrar que o erro máximo absoluto para a média, com 100(1 - α)% de confiança, é dado por:

$$|E| = t_{\alpha/2,b-1}\sqrt{\frac{V_r}{b}} \tag{4}$$

, em que $t_{\alpha/2,b-1}$ é o ponto percentual *1 - α/2* da distribuição *t* com *b - 1* graus de liberdade.

O processo de avaliação é baseado nas equações acima e leva em consideração os parâmetros definidos pelo projetista através das anotações no código do tipo: <$ *NConf* | *EMax* | *TMax* $> (ver linha 1 do código da Seção 8.1), onde *NConf*, *EMax* e *TMax* definem o nível de confiança, o erro máximo absoluto para energia e o erro máximo absoluto para o tempo, respectivamente. O processo de avaliação segue gerando novas replicações (inicialmente são dez) até que o erro máximo absoluto para a energia e para o tempo de execução, calculados pela a Equação 4, sejam menores ou iguais àqueles definidos pelo projetista.

7 Metodologia de avaliação

O processo de avaliação de um código segue a metodologia exibida na Figura 7. Primeiro, o código anotado a ser avaliado é compilado por um Assembler. Em seguida, o código binário gerado no primeiro passo é utilizado pelo Compilador Binário-CPN para gerar a composição dos modelos básicos. O Compilador Binário-CPN gera dois formatos de saída: um compatível com o CPN Tools e outro compatível com o ALUPAS. O CPN Tools é um *software* para edição e simulação de CPN, sendo este utilizado para validar os modelos gerados pelo Compilador Binário-CPN. O último passo é avaliar o modelo gerado pelo Compilador Binário-CPN e apresentar os resultados.

Devido às limitações de desempenho apresentadas pelo CPN Tools ao avaliar os modelos, um simulador específico foi criado. O algoritmo [10] de simulação é composto pelas

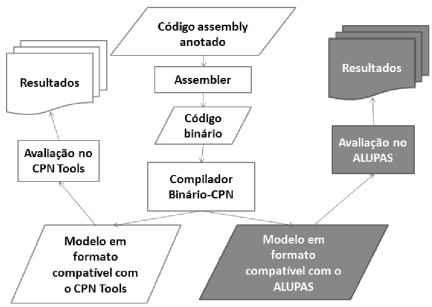

Figura 7. Processo de avaliação no ALUPAS.

seguintes componentes: (i) Estado da simulação: conjunto de variáveis necessárias para descrever o estado da simulação em um tempo particular, que no ALUPAS é representado pela marcação da rede; (ii) Tempo da simulação: variável que representa o valor atual do tempo simulado; (iii) Agenda de eventos: lista contendo o tempo de simulação em que cada transição do modelo estará habilitada; (iv) Contadores estatísticos: variáveis utilizadas para alocar informações estatísticas sobre as métricas a serem avaliadas; (v) Critério de parada: são criadas várias replicações da simulação até que o critério de parada seja satisfeito (ver Seção 6).

A Figura 8 descreve o algoritmo de simulação do ALUPAS. A interface de entrada do ALUPAS é exibida na Figura 9. Para avaliar um código, o projetista pressiona *Compile* (gera o modelo CPN), e em seguida, pressiona *Simulate*. A Figura 10 exibe a tela de resultados do ALUPAS.

8 Experimentos e resultados

Alguns experimentos foram realizados a fim de demonstrar e validar o funcionamento do ALUPAS. Todos os experimentos foram feitos utilizando uma máquina com a seguinte configuração: Pentium IV HT 3Ghz, 1.5 Gb RAM e sistema operacional Windows XP.

8.1 Experimento 1

O primeiro experimento consiste em avaliar o código assembly do algoritmo de Busca Binária para um vetor com 255 elementos:

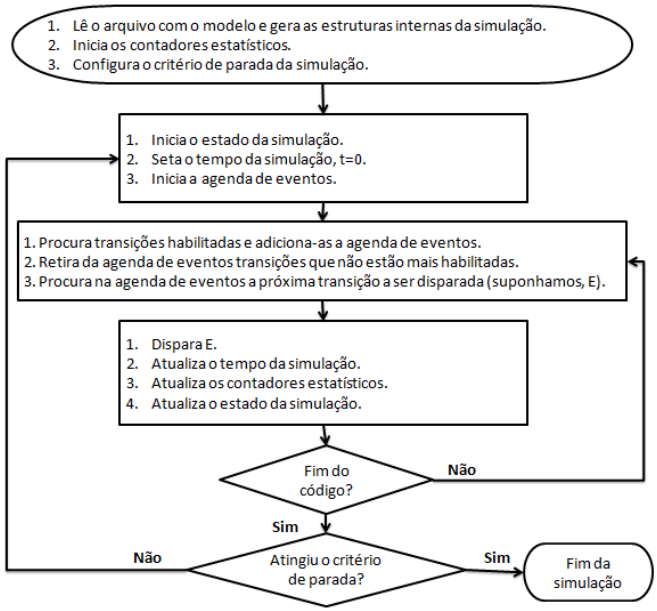

Figura 8. Algoritmo de simulação.

```
1.  ;<$0.95|0.1|0.1>
2.        main MOV   r0,#0
3.             MOV   r1,#0xfe
4.             MVN   r3,#0
5.             B     loop
6.
7.       begin ADD   r12,r0,r1
8.             ASR   r2,r12,#1
9.             LDR   r12,|L1.4084|
10.            LDR   r12,[r12,r2,LSL #3]
11.            CMP   r12,#0x12c
12.            BNE   notfound                ;<@1.0@>
13.            LDR   r12,|L1.4084|
14.            ADD   r12,r12,r2,LSL #3
15.            LDR   r13,[r12,#4]
16.            B     end
17.
```

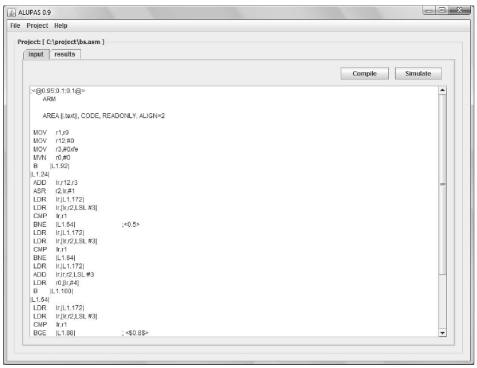

Figura 9. Interface de entrada do ALUPAS.

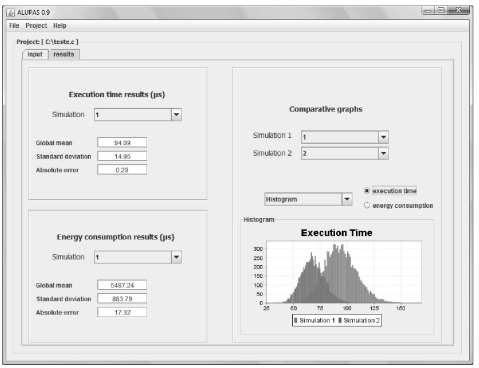

Figura 10. Interface de saída do ALUPAS.

```
18.  notfound LDR   r12,|L1.4084|
19.           LDR   r12,[r12,r2,LSL #3]
20.           CMP   r12,#0x12c
21.           BGE   inf                    ;<@0.0>
22.           ADD   r0,r2,#1
23.           B     loop
24.
25.       inf SUB   r1,r2,#1
26.
27.      loop CMP   r0,r1
28.           BLE   begin                  ;<@0.88@>
29.
30.       end MOV   r0,r0
31.           BX    lr
```

Para um melhor entendimento do código, seu equivalente na linguagem C é exibido abaixo:

```
1. int   fvalue, mid, up, low;
2.
3.  low = 0;
4.  up = 254;
5.  fvalue = -1 /* all data are positive */ ;
6.  while (low <= up) {
```

```
7.      mid = (low + up) >> 1;
8.      if (data[mid].key == x) { /* found  */
9.         fvalue = data[mid].value;
10.        break;
11.     } else
12.     if (data[mid].key < x) { /* not found */
13.        low = mid + 1;
14.     } else {
15.        up = mid - 1;
16.     }
17. }
```

O comportamento da execução do código assembly depende do elemento buscado (chave) e dos elementos presentes no vetor. Esta variação é definida pelas instruções de desvio condicional: Inst12 (linha 12) e Inst21 (linha 21) e Inst28 (linha 28). Durante a execução, o fim das iterações é definido por Inst28 ou Inst12. Se o elemento for localizado em uma determinada iteração, Inst12 não realiza o desvio e a busca termina com sucesso nesta iteração. Se o elemento buscado não for encontrado em nenhuma das iterações, Inst28 é quem limita o número máximo de iterações. Inst21 define se a busca deve continuar na metade posterior ou inferior do vetor.

O código foi avaliado em três cenários diferentes: melhor caso (BCET - *Best Case Execution Time*), caso médio (TCET - *Typical Case Execution Time*) e pior caso (WCET - *Worst Case Execution Time*). Para cada cenário foi associado às instruções condicionais um conjunto específico de probabilidades. Estas probabilidades representam o fluxo de execução do código para o cenário escolhido.

O pior caso ocorre na busca de um elemento inexistente. No pior caso, o número de iterações é 8 ($\log_2(255)$). Em contrapartida a este cenário, no melhor caso, o elemento é encontrado na primeira iteração, isto é, no meio do vetor. No caso típico, a probabilidade de um elemento ser localizado na i-ésima iteração é $\frac{2^{i-1}}{255}$. Assim, o número médio de iterações necessárias para localizar um elemento é $\sum_{i=1}^{8} i \cdot \frac{2^{i-1}}{255} \cong 7$.

No pior caso, Inst28 é quem delimita o fim das iterações. Para que sejam executadas 8 iterações, é necessário que a condição utilizada por Inst28 seja avaliada como verdadeira 8 vezes. Assim, probabilidade de Inst28 é $p = \frac{8}{9}$. Sabendo que a chave não deverá ser encontrada no vetor, a probabilidade de Inst12 é "1". Adicionalmente, no pior caso, o desvio definido por Inst14 nunca é efetuado, isso ocorre quando o elemento buscado é maior que qualquer elemento presente no vetor. Assim, a probabilidade de Inst14 é "0".

No melhor caso, só há uma iteração. O desvio especificado por Inst28 sempre ocorre e logo em seguida o elemento é identificado por Inst12. Desta forma, para avaliar o melhor

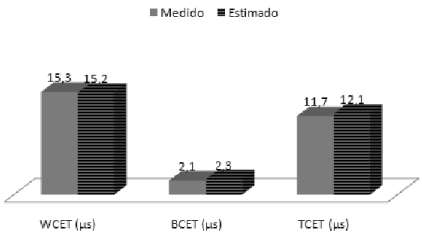

Figura 11. Tempo de execução: estimado x medido.

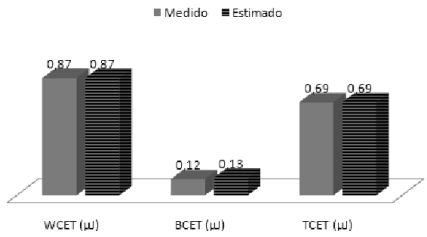

Figura 12. Consumo de energia: estimado x medido.

caso, as probabilidades das instruções Inst28 e Inst12 devem ser "1" e "0", respectivamente. A probabilidade de Inst21 é irrelevante, já que no melhor caso, o fluxo nunca chegará a este ponto.

No caso típico, quem delimita o fim das iterações é Inst12. Assim, Inst28 tem probabilidade "1". Havendo em média 7 iterações, Inst12 tem probabilidade "$\frac{6}{7}$" de realizar o desvio. Isto é, na sétima verificação Inst12 não realiza o desvio e a busca termina com sucesso. No caso típico, Inst21 tem probabilidade "0.5", já que a busca possui igual probabilidade de continuar na metade posterior ou inferior do vetor.

Para cada cenário, Figura 11 e Figura 12 exibem os resultados reais medidos no *hardware* e as estimativas geradas por ALUPAS. Analisando as figuras, é possível observar que as estimativas foram bastante exatas. O erro entre o que foi medido e estimado foi em média de 3%. Para obter do *hardware* os resultados do melhor caso, a execução de uma busca por um elemento que estava no meio do vetor foi medida. A mesma abordagem foi aplicada para obter os resultados do pior caso, no entanto, a chave do elemento buscado era maior que a chave de todos os elementos presentes no vetor. Os valores do caso típico, foram obtidos através dos valores médios de 50 execuções do Busca Binária, onde os elementos buscados eram elementos aleatórios (distribuição uniforme) no vetor.

8.2 Experimento 2

O segundo experimento tem como objetivo fazer um comparativo entre a velocidade de simulação do ALUPAS e o CPN Tools. Ele também demonstra a importância do processo de redução do modelo. Este experimento é formado por códigos que utilizam apenas o modelo de instrução ordinária (ver Figura 2). O tamanho dos códigos avaliados varia em ordem crescente de 10 a 400 instruções.

Tabela 1. Comparativo do tempo de simulação.

	Modelo CPN		Modelo CPN reduzido	
N Inst.	CPN Tools	ALUPAS	CPN Tools	ALUPAS
10	17s	187ms	6s	187ms
20	21s	219ms	6s	187ms
30	30s	234ms	6s	187ms
40	41s	281ms	6s	187ms
50	56s	297ms	6s	187ms
100	1min 51s	515ms	6s	187ms
200	5min 6s	1s 219ms	6s	187ms
400	17min 25s	3s 438ms	6s	187ms

Os códigos foram simulados de quatro formas diferentes: i) Usando o CPN Tools com modelo CPN não-reduzido; ii) Usando CPN Tools e o modelo CPN reduzido; iii) Usando ALUPAS com o modelo CPN não-reduzido; iv) Utilizando o ALUPAS com o modelo CPN reduzido. A Tabela 1 exibe o comparativo dos tempos de simulação dos modelos, onde se observa que o ALUPAS foi entre 32 e 303 vezes mais rápido que o CPN Tools. Essa diferença aumenta à medida que o tamanho do modelo cresce. Também, é possível verificar a importância do processo de redução e que as simulações utilizando o modelo CPN reduzido apresentaram tempo de simulação constante. Isto é explicado pelo fato de os códigos simulados não apresentarem instruções de desvio condicional. Este padrão permitiu que o processo de redução agrupasse as instruções do código em apenas um bloco básico.

8.3 Experimento 3

O terceiro experimento objetiva validar o ALUPAS através da avaliação de uma aplicação embarcada real: o Oxímetro de Pulso [9]. O oxímetro é um equipamento responsável por medir o nível de saturação do oxigênio de um paciente utilizando um método não invasivo. Esta aplicação utiliza duas categorias tecnológicas distintas: a tecnologia analógica e a digital. A aquisição e condicionamento do sinal proveniente do meio biológico é feita por estruturas analógicas devido à natureza analógica do sinal de interesse. A extração da informação de saturação e freqüência cardíaca é realizada mediante a digitalização do sinal e seu processamento por algoritmos computacionais específicos. Sendo assim, o instrumento é formado por um módulo analógico e um microprocessado (digital). A unidade microprocessada cumpre o papel de controlar os padrões de excitação de forma a manter a melhor situação de leitura possível. A interface com o usuário é constituída de mostradores de sete segmentos e chaves de programação de alarmes máximos e mínimos para a saturação e freqüência car-

díaca. Alguns oxímetros oferecem também a apresentação do sinal fotopletismográfico em um mostrador gráfico.

O sistema do oxímetro de pulso é dividido em três partes distintas, denominadas planos. Os planos são procedimentos que executam independentemente um dos outros. Cada um destes planos é dividido em sub-tarefas. Seguindo este conceito, nomeia-se os planos de acordo com a atividade macro que eles desempenham. São elas:

- Plano 1 - Rotina de Excitação;
- Plano 2 - Rotinas de Amostragem e Controle;
- Plano 3 - Programa Gerenciador.

Neste estudo de caso, considerou-se para avaliação apenas o Plano 1 e o Plano 2, dado que o Plano 3 diz respeito à inicialização do sistema, atendimento aos comandos do usuário entre outras atividades de menor relevância neste contexto.

No Plano 1 - Rotina de Excitação são realizadas as seguintes tarefas:

- Geração dos pulsos de excitação, vermelho e infravermelho;
- Temporização entre os pulsos de excitação vermelho e infravermelho;
- Controle da freqüência dos pulsos de excitação: vermelho e infravermelho;
- Leitura do conversor digital/analógico.

O Plano 2 - Rotinas de Amostragem e Controle é composto por dois procedimentos. Estes dois procedimentos são executados em seqüencia e realizam as seguintes tarefas:

- Amostragem
  - Disparo de Conversão A/D;
  - Aguarda conversão;
  - Lê conversor;
  - Carrega vetores;
- Controle
  - Regula varredura;
  - Mostra resultados/curva.

- Gerenciamento;
- Cardiobeep.

Foram avaliados os casos médios de execução dos Planos 1 e 2. Fluxos de exceção que ocorrem, por exemplo, quando a aplicação está calibrando o circuito de excitação ou quando são detectados erros no recebimento dos sinais de entrada, não foram considerados. A Tabela 2 exibe os resultados do experimento.

Tabela 2. Resultados do exemplo 5 - Oxímetro de Pulso.

	Tempo de execução		Consumo de energia		Erro: estimado x medido	
Plano	Estimado	Medido	Estimado	Medido	Energia	Tempo
Excitação	38.89 μs	38.88 μs	2.29 μJ	2.25 μJ	1.8%	0.02%
Amostragem	86.61 μs	91.18 μs	5.16 μJ	5.55 μJ	7.5%	5.27%
Controle	12410.78 μs	12745.99 μs	708.59 μJ	779.46 μJ	10%	0.27%

9 Conclusão

Este trabalho apresentou o ALUPAS, um simulador estocástico cujo objetivo é estimar o desempenho e consumo de energia em *softwares* para sistemas embarcados. A modelagem da simulação foi feita através uma abordagem formal utilizando as Redes de Petri Coloridas (CPN), onde cada instrução de um microcontrolador da família ARM7 foi modelada de forma a reproduzir seu comportamento. Utilizando uma abordagem estocástica, mostrou-se que os vários cenários de execução do *software* podem ser analisados de maneira simples. Os resultados dos experimentos mostraram-se bastante exatos em comparação aos valores reais medidos no *hardware*, além disso, ALUPAS apresentou desempenho bem superior ao CPN Tools.

Uma vez que as CPN fornecem o formalismo ideal para modelar sistemas concorrentes, paralelos e assíncronos, uma extensão natural deste trabalho é estender os modelos atuais para que seja possível a análise em sistemas embarcados multiprocessados (múltiplos processadores). Neste contexto, outras questões devem ser levadas em consideração no modelo de simulação como, por exemplo, a comunicação inter-processador.

Referências

[1] Arm7tdmi technical reference manual. *ARM DDI 0210A*, 2001.

[2] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. Emma, and M. Rosenfield. Microarchitecture-level power-performance analysis: the powertimer approach. *IBM J. Research and Development*, 47(5/6):653–670, 2003.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, 2000.

[4] W. W. Hines, D. C. Montgomery, D. M. Goldsman, and C. M. Borror. *Probabilidade e Estatística na Engenharia*. 2006.

[5] C. Huang, B. Zhang, A. Deng, and B. Swirski. The design and implementation of PowerMill. *Proceedings of the 1995 international symposium on Low power design*, pages 105–110, 1995.

[6] M. Irwin, M. Kandemir, and N. Vijaykrishnan. SimplePower: A Cycle-Accurate Energy Simulator. *IEEE CS Technical Committee on Computer Architecture Newsletter*, 2001.

[7] R. Jayaseelan, T. Mitra, and X. Li. Estimating the worst-case execution energy of embedded software. *Proceedings of the Twelfth Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 81–90, Apr. 2006.

[8] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 1 of *EATCS Monographs in Computer Science*. Springer-Verlag, 1992.

[9] M. N. O. Junior. Desenvolvimento de um protótipo para a medida não invasiva da saturação arterial de oxigênio em humanos - oxímetro de pulso. Master's thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco, Agosto 1998.

[10] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 1997.

[11] Mentor. http://www.mentor.com/.

[12] N. Nikolaidis and P. Neofotistos. Instruction-level Power Measurement Methodology. *Electronics Lab, Physics Dept., Aristotle University of Thessaloniki, Greece, March*, 2002.

[13] M. Oliveira, S. Neto, P. Maciel, R. Lima, A. Ribeiro, R. Barreto, E. Tavares, and F. Braga. Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured petri nets. *ICATPN 2006, LNCS*, 4024:261–281, 2006.

[14] A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. *Applications and Theory of Petri Nets*, 2003:24th, 2003.

[15] A. Sinha and A. Chandrakasan. JouleTrack-A Web Based Tool for Software Energy Profiling. *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 220–225, 2001.

[16] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An accurate and fine grain instruction-level energy model supporting software optimizations. *Proc. of PATMOS*, 2001.